Reducing Software Security Risk through an Integrated Approach

David P. Gilliam
Caltech, Jet Propulsion Laboratory
david.p.Gilliam@jpl.nasa.gov

John C. Kelly
Caltech, Jet Propulsion Laboratory
john.c.kellyw@jpl.nasa.gov

John D. Powell
Caltech, Jet Propulsion Laboratory
John.Powell@jpl.nasa.gov

Matt Bishop
University of California at Davis
bishop@cs.ucdavis.edu

Abstract

This paper presents joint work by the California Institute of Technology's Jet Propulsion Laboratory and the University of California at Davis (UC Davis) sponsored by the National Aeronautics and Space Administration Goddard Independent Verification and Validation Facility to develop a security assessment instrument for the software development and maintenance life cycle.

Vulnerabilities in operating systems and software applications render an otherwise secure environment insecure. Any operating system or application added to a secure environment that has exploitable security vulnerabilities affects the security of the whole environment. An otherwise secure system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities. Therefore, it is critical that software on networked computer systems be free from security vulnerabilities.

Security vulnerabilities in software arise from a number of development factors; but these vulnerabilities can generally be traced to poor software development practices, new modes of attacks, mis-configurations, and unsecured links between systems.

A Software security assessment instrument can aid in providing a greater level of assurance that software is not exposed to vulnerabilities as a result of defective software requirements, designs, code or exposures due to code complexity and integration with other applications that are network aware.

This paper presents research on the generation of a software security assessment instrument to aid developers in assessing and assuring the security of software in the development and maintenance lifecycles. The research presented here is available at: http://security.jpl.nasa.gov/rssr.

Keywords

Security Toolset, Vulnerability Matrix, Property-Based Testing, Model Checking, Security, Verification

1. Introduction

Software on networked computer systems must be free from security vulnerabilities. Security vulnerabilities in software arise from a number of development factors that can generally be traced to poor software development practices, new modes of attacks, mis-configurations, and unsecured links between systems. An otherwise secure system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities.

Currently, there is a lack of Security Assessment Tools (SATs) for use in the software development and maintenance life cycle to mitigate these vulnerabilities. The National Aeronautics and Space Administration (NASA) has funded the Jet Propulsion Lab in conjunction with the University of California at Davis (UC Davis) to develop a software security assessment for use in the software development and maintenance life cycle.

The goal of the effort is the use of a formal analytical approach for integrating security into existing and emerging practices for developing high quality software and computer systems. The approach is to develop a security assessment instrument consisting of a collection of tools, procedures and instruments to support the development of secure software. Specifically, the instrument offers a formal approach for engineering network security into software systems and application throughout the software development and maintenance life cycles.

The security assessment instrument has three primary foci: a Vulnerability Matrix (VM), a collection of

Security Assessment Tools (SAT) which includes the development of a Property-Based Testing (PBT) tool, and a Model-Based Verification (MBV) instrument.

The VM is a database maintained by UC Davis as part of the Database of Vulnerabilities, Exploits, and Signatures (DOVES) project. It contains a list of vulnerabilities, the associated platform/application, and the exploit signature fields.

The VM provides a searchable knowledge base from which properties may be extrapolated for use with PBT and MBV. This knowledge base can also accommodate the discovery of new attacks not yet seen on the internet, but which may be discovered through MBV techniques.

The SAT is a collection of tools and programs that can be used to check the security of software requirements, designs and source code. Each of the SAT's includes a description of the tool and it use, its pros and cons, related tools, and where the particular tool can be obtained.

As part of the SAT, UC Davis is developing from a prototype a PBT tool. This PBT will slice software code looking for specific vulnerability properties. Property based testing is a tool that verifies properties against the code level implementation of a system. These properties are extracted from the VM, which may have grown due to properties being added through the use of MBV. Additionally, PBT is equipped with its own libraries that contain readily testable properties. Finally, used with the MBV, the PBT can provide verification of a model's fidelity to the system in the MBV.

The MBV component of the research is a operational approach to perform verification of software designs for compliance to security properties. The *Flexible Modeling Framework* (FMF) approach is an innovative model checking approach that will facilitate the development and verification of software security models as composable components

Model based verification uses precise abstractions. It offers the ability to verify security properties over system models early in the life cycle – before an implementation exists. MBV can effectively identify security anomalies that have not been discovered as a result of a known network security attack. These new anomalies may then be added to the Vmatrix Anomalies that are found in early lifecycle phases through the examination of abstractions (models) can be preserved and later passed on to the PBT for verification at the code level.

The inception of this work was previously reported to IEEE WETICE Workshop on Enterprise Security.[1], 4th Annual Assurance Technology Conference at Glenn Research Center and the NASA OSMA Software Assurance Symposium '01 sponsored by the NASA Goddard IV&V Facility. Three parts have been accomplished to date, the Vulnerability Matrix (Vmatrix), the initial collection of Security Assessment Tools (SATs), and the Property Based-Testing (PBT) instrument. A fourth part, the Model-Based Verification (MBV) instrument will be completed in April, 2002.

Assessments of high profile NASA systems believed to be vulnerable to attack will provide a metric to determine the effectiveness of these activities and prototypes. The security assessment instrument will be verified on a JPL/NASA Class A Flight Project to assess the approach and the viability of the security assessment instrument for assuring the security of software on critical networked systems.

2. Vulnerability Matrix (Vmatrix)

The VMatrix task was initiated to develop a searchable database containing a taxonomy of vulnerabilities and exposures and to catalogue them into libraries of properties that can be used in conjunction with the PBT and MBV instruments to assess the security of software code to assure that the software is free from the specified vulnerabilities and exposures. Of particular concern is that the properties of these vulnerabilities and exposures are not re-introduced during integration with operating systems or interoperability with other applications, nor in the introduction of upgrades to either the operating system or applications running on them.

Additionally, the information in the database is intended, in part, to provide network security professionals an understanding of the vulnerabilities and their exploits so they can better secure their systems. Equally important, it also provides developers with an understanding of the vulnerabilities and exposures in code that introduce security risks to software and systems. The intended goal is to enable developers to write more secure code and to provide a greater level of assurance that software code is not exposed to vulnerabilities when integrated with systems and other applications when used in a networked environment.

The Vmatrix, examines vulnerabilities and exposures and the methods used to exploit them. The VMatrix lists vulnerabilities and exposures along with their Common Vulnerabilities and Exposures (CVE) listing[2]. The VMatrix includes a brief summary and a description of the vulnerability or exposure, the affected software or operating system, how to detect the vulnerability or exposure and the fix or method for protecting against the exploit. Also included is catalogue information, keywords, and other related information as available, regarding the vulnerability or exposure. Interesting links, including links to Mitre with the CVE listing and the Ernst and Young website where vulnerabilities and exposures are ranked by severity and frequency among other factors, are also provided.

The VMatrix led to the development and extension of a database controlled and maintained by UC Davis, the

Database of Vulnerabilities, Exploits, and Signatures, (DOVES). DOVES contains additional vulnerabilities and exposures beyond that which is now contained in the VMatrix.

The Vmatrix, the DOVES database along with the SATs (discussed below) are available from websites at JPL and UC Davis which can be reached from: http://security.jpl.nasa.gov/rssr.

3. Security Assessment Tools (SATs)

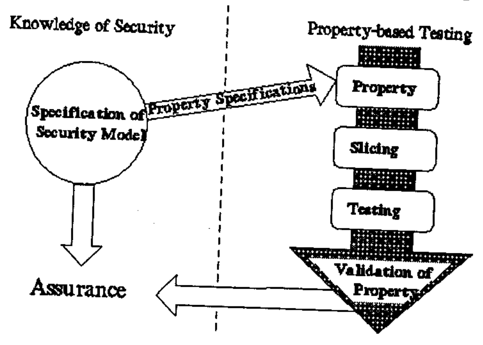
The Security Assessment Tools are free tools that have been developed and collected for use in testing and assuring the security of operating systems and software. This collection is provided as a list on the web sites noted above. The SATs are a listing of tools that contain a brief summary stating the purpose of the tool, where the tool can be obtained, and their use along with pros and cons of each of the tools. Also provided, is a list of similar tools or alternative tools, and a classification of each tool. A journal paper, "A Classification Scheme for Security Tools," provided on the SATs web page, discusses a classification scheme of these security related tools and their usage.

A more complete description of the tools and a discussion of how to use each of the tools is currently being developed. Additional SATs are being collected as they become available to include in the current list.

The SATs will be categorized and cross-referenced to alternate tools so that code developers, system administrators, and network and computer security professionals can have a central location to search for specific tools for use in writing secure software code and securing computer systems.

4. Property-Based Testing

The role of property-based testing is to bridge the gap between formal verification and ad hoc verification. This provides a basis for analyzing software without sacrificing usefulness for rigor, yet capturing the essential ideas of formal verification. It also allows a security model to guide the testing for security problems
Property-based testing [3] is a technique for testing that programs meet given specifications. The tester gives the specifications in a language that ties the specification to particular segments of code. The specification has assertions, which indicate changes in the security state of the program, and properties, which describe a specific set of states that are considered secure in this context. The idea is to ensure that the properties always hold.
The tester consists of two parts. The *instrumenter* inserts statements into the source code that emit assertions about the current state of execution. The *execution monitor* takes that information as input and determines if the current state of execution violates any of the properties. If so, the program has a security flaw. The instrumenter, execution monitor, and any libraries of desireable security properties make up the Tester's Assistant (TA).[4]



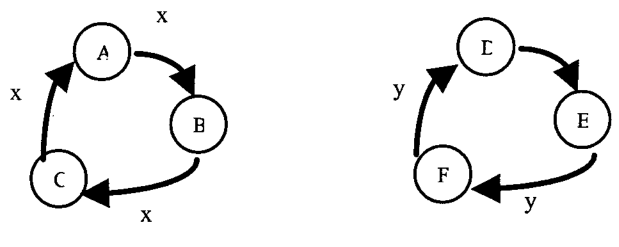
PBT Model
Figure 1

Our goal was to develop the TA to test programs written in C++ code for the UNIX environment. However, the TA task has been changed to test programs written in JAVA instead. This eliminates some problems such as pointer aliasing (because JAVA does not have it). It also introduces some problems, because certain system functions (such as the printing functions) are not written in JAVA. If the call to such a function is instrumented, the native code instrumented, or the statements must surround the call to the routine instead of being invoked as the first instruction in the routine. The first would require developing a much more general instrumenting tool, so we opt for the second. When the method being invoked is computed at runtime, the complexity of the wrapping instatements is considerable,

We have also modified the TASPEC specification language[5] to clarify ambiguities uncovered by our testing. For example, consider the assertions authenticated (bob), password (bob), password (alice) are present in the database. The instrumented program puts out the property authenticated (x) and password (x). Does the execution monitor report a violation, because there exists one value for x such that the property fails, or does it say the property is satisfied, because there exists one value of x such that the property holds? We have chosen the latter, but one could equally well choose the former. The only difference that would cause is in the writing of specifications.

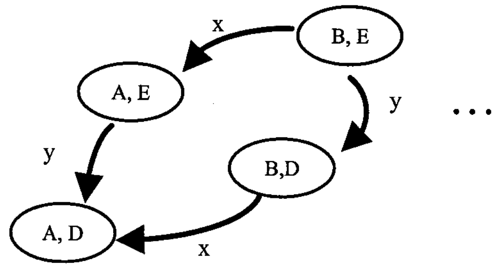
5. Model-Based Security Specification and Verification

Model based specification and verification make use of discrete finite models to verify compliance of the model to desired properties; in this case, software/network security properties. Network security properties often focus on characteristics that are manifested though the operation of multiple software applications and systems operating concurrently with an attacking process. The concurrent nature of the systems results in an operational space that is too large to effectively verify security properties through traditional testing of the implementation. Further, vulnerabilities introduced in the early phases of the development lifecycle are difficult or impossible to remove in later phases when an implementation is being tested. This results in the addition of cumbersome workarounds and "patches" to secure the software system. Model based verification offers the opportunity to verify properties early in the life cycle, providing a clearer understanding of the vulnerability issues within the system before an implementation exists.



Process P1 Process P2

Figure 2



Processors P1, P2
Figure 3

Model checkers automatically explore all paths in a finite state space from a given start state in a computational tree. The objective is to verify system properties over all possible scenarios within a model. Model Checkers differ from more traditional heavyweight formal techniques in that:

- Model checkers are operational as opposed to deductive
- Model checkers provide counter examples when properties are violated (counter examples)
- Their goal is oriented toward finding errors as opposed to proving correctness since the model is an abstraction of the actual system

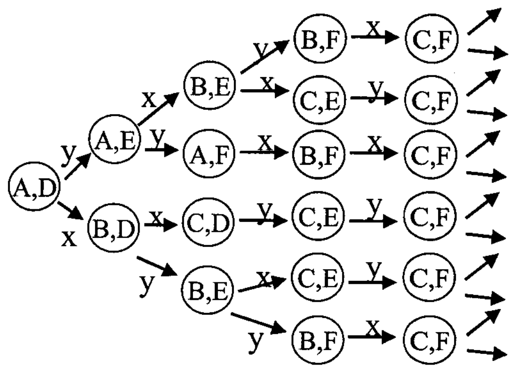


Figure 4

Model based verification techniques, such Model Checking, are not without drawbacks. Among them are the ability to model a system with a high degree of fidelity in a timely manner while the system evolves. This is particularly problematic in the earliest stage of development such as requirements and high-level design when the system definition is most volatile. This lack of agility limits an analysts ability to maintain an up to date model that and minimize the latency between the introduction of errors and their discovery.

A limitation specific to model checking is the state space explosion problem. Similar to the growth of the growth of the operational space mentioned above, the state space that a model checker must search to verify properties grows at an exponential rate as the model becomes more detailed. As shown in figures 2 through 4 the state space grows at a rate of m^n where m is the range of possible values a variable may assume and n is the number of variables in the model. Despite the use of modeling techniques such as abstraction and homomorphic reduction it is infeasible to verify all but the most simplistic software systems in their entirety though model checking.

An innovative verification approach that employs model checking as its core technology is offered as a means to bring software security issues under formal control early in the life cycle while mitigating the drawbacks discussed above. The *Flexible Modeling Framework* (FMF) is an approach that employs:

- A system for building models in a component based manner to cope with system evolution in a timely manner
- A compositional verification approach to delay the effects of state space explosion and allow property verification results to be examined with

respect to larger, complex models in an indirect manner.

Modeling in a component-based manner involves the building of a series of small model, which will later be strategically combined for system verification purposes. This correlates the modeling function with modern software engineering and architecture practices where by a system is divided into major parts, and subsequently into smaller detailed parts, and then integrated to build up a software system. An initial series of simple components can be built when few operational specifics are known about the system. However these components can be combined and verified for consistency with properties of interest such as software security properties. As the system evolves only the affected components need be modified. Further by retaining knowledge from previous verifications the effort of re-verifying properties may be reduced significantly. This will result in a decreased cycle time for verification of model updates thus improving the timeliness of the formal verification results. As more is learned about the system's specific manner of accomplishing its task(s) the affected model components can be:

- Modified to reflect the more detailed approaches developed during the design phase.
- Segmented into its own series of components when the complexity of the high level component begins to exhibit state space explosion problems.

The approach of compositional verification used in the FMF seeks to verify properties over individual model components and then over strategic combinations of them. The goals of this approach are to: 1) infer verification results over systems that are otherwise to large and complex for model checking from results of strategic subsets (combinations) while minimizing false reports of defects. 2) Retain verification results from individual components and combination to increase the efficiency subsequent verifications and ultimately aid in the strategic combination selection process. The FMF verification process begins determining which model components are safe and unsafe with respect to the property in question. Then, the strategic combination process seeks to build up relationships between components. Figure 5 shows an example where the components C_1 and C_3 are safe with respect to some security property while the states C_2 and C_4 are unsafe. Relationships between C_1 and C_2 as well as C_3 and C_4 are shown. Since C2 is individually unsafe, C_1 is individually safe and the combination C_1 and C_2 is safe, C_1 is said to *mitigate* C_2. Conversely C3 is safe and C4 is unsafe and the combination of the two components is unsafe. In this case C_4 is said to *undermine* C_3. It bears noting that two

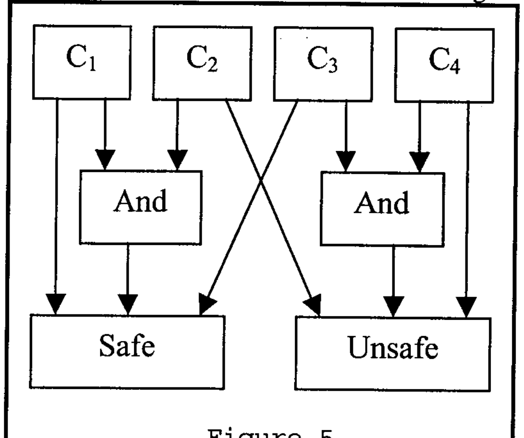


Figure 5

components that are labeled individually safe may produce and unsafe security condition when combined and vise versa.

Maintaining the network of relationships for each property will allow future verifications of the property to be accomplished by noting the relationships that were used to make earlier verification inferences and only re-verifying the relationships affected by a component change or addition.

This approach is currently under development and shows promise for early life cycle detection of security vulnerabilities. The approach may be generalized and/or tailored in future work for applicability to non-security domains such as safety.

6. Instrument Integration

The various parts of the Security Assessment Instrument can be used separately or in combination (See Figure 6) providing the additional benefits of:

- Reduced rework to identify security properties
- Increased confidence in the system through verification at multiple times during the development and maintenance lifecycle
- One tool is capable of verifying the input and output of another tool in the instrument
- Finding additional attacks yet to be seen in the wild (attacks that have not yet been seen outside of a laboratory environment) and test for their viability and severity

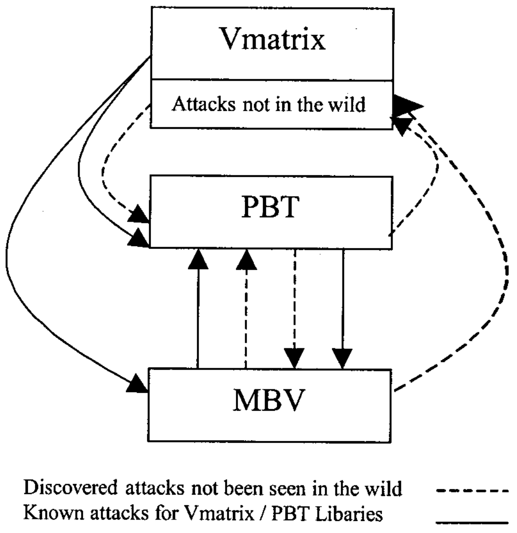

Vmatrix
Attacks not in the wild
PBT
MBV
Discovered attacks not been seen in the wild
Known attacks for Vmatrix / PBT Libaries

Figure 6

6.1. Vulnerability Matrix (VMatrix)
The vulnerability matrix provides a searchable knowledge base from which properties may be extrapolated for use with PBT (See Section 6.1.) and Model Based Verification (MBV) (See Section 6.3.). This knowledge base can also accommodate the discovery of new attacks not yet seen in the wild that may be discovered through MBV techniques.

6.2. Property Based Testing (PBT)
Property based testing is a tool that verifies properties against the code level implementation of a system. These properties are extracted from the VMatrix (See Section 6.1.), which may have grown due to properties being added through the use of MBV (See Sec 6.3.). Additionally, PBT is equipped with its own libraries that contain readily testable properties. Finally, used with the MBV, the PBT can provide verification of a model's fidelity to the system in the MBV.

6.3. Model Based Verification (MBV)
Due to the fact that Model based verification uses precise abstractions; it offers the ability to verify security properties early in the life cycle – before an implementation exists. The MBV can effectively identify and notify the VMatrix of security anomalies that are not yet seen in the wild (See Sec 6.1.). Anomalies found early in the lifecycle by examining abstractions can later be passed on to the PBT for verification at the code level (See Sec 6.2.).

7. Conclusion

The four parts of the integrated approach for detecting security vulnerabilities in software form a coherent technique for examining systems for software security flaws. Each part can be used independently or in conjunction with another. When used in conjunction with each other, synergistic benefits are leveraged to classify and understand security properties for modeling and testing. The VMatrix and model-based checking provide the properties that the software must meet; the property-based tester checks that implementations do indeed meet these properties. The VMatrix forms the beginning of a library of properties. Property-based testing requires properties expressed in TASPEC to test against. Training in the writing of more secure programs flows directly from the library of security properties. Placing these properties in the context of a particular system environment is an important part of improving the quality of software and systems.

8. Acknowledgements

The research described in this paper is being carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration, and the University of California at Davis under a subcontract with the Jet Propulsion Laboratory, California Institute of Technology.

9. References

[1] D. Gilliam, J. Kelly, M. Bishop, "Reducing Software Security Risk Through an Integrated Approach," Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June, 2000), Gaithersburg, MD, pp.141-146.

[2] Published and maintained by Mitre. The CVE listing can be found at: http://cve.mitre.org/

[3] G. Fink, M. Bishop, "Property Based Testing: A New Approach to Testing for Assurance," *ACM SIGSOFT Software Engineering Notes* **22**(4) (July 1997).

[4] M. Bishop, "Vulnerabilities Analysis," *Proceedings of the Recent Advances in Intrusion Detection* (Sep. 1999).

[5] J. Dodson, "Specification and Classification of Generic Security Flaws for the Tester's Assistant

Library," M.S. Thesis, Department of Computer Science, University of California at Davis, Davis CA (June 1996).

[6] J. R. Callahan, S. M. Easterbrook and T. L. Montgomery, "Generating Test Oracles via Model Checking," NASA/WVU Software Research Lab, Fairmont, WV, Technical Report # NASA-IVV-98-015, 1998.

[7] P. E. Ammann, P. E. Black and W. Majurski. "Using Model Checking to Generate Test Specifications," 2nd International Conference on Formal Engineering Methods (1998) pp. 46-54.

[8]G. Lowe. Breaking and Fixing the Needham-Schroeder Public Key Protocol Using CSP and FDR. In TACAS96, 1996.

[9] W. Wen and F Mizoguchi. Model checking Security Protocols: A Case Study Using SPIN, IMC Technical Report, November, 1998.